EFFICIENT SWITCHING BETWEEN WIFI AND CELLULAR NETWORKS FOR ROBUST INTERNET CONNECTIVITY

by

Naveen Dasa Subramanyam

A thesis submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Master of Science

in

Computing

School of Computing

The University of Utah

August 2014

The University of Utah Graduate School

STATEMENT OF THESIS APPROVAL

The thesis of **Naveen Dasa Subramanyam**

has been approved by the following supervisory committee members:

Sneha Kumar Kasera	, Chair	**05/01/2014** Date Approved
Jacobus Van der Merwe	, Member	**05/01/2014** Date Approved
Robert Ricci	, Member	**05/01/2014** Date Approved

and by **Ross Whitaker** , Chair of

the School of **Computing**

and by David B. Kieda, Dean of The Graduate School.

ABSTRACT

Current solutions for switching between cellular networks and WiFi, such as those provided by Google and Samsung, do not work well in many scenarios. For example, a Samsung Galaxy S4 smartphone connected to a WiFi network remains in the WiFi mode for more than 5 minutes, even when the Internet connectivity beyond the first hop is unavailable. Therefore, there is a strong need for an efficient and automated switching mechanism between WiFi and cellular networks that has wide applicability.

We design an adaptive, energy-efficient methodology to switch between WiFi and cellular networks to provide robust Internet connectivity. When using a WiFi network, we utilize a combination of active and passive monitoring techniques to detect unavailability of Internet connectivity via WiFi. When using the cellular network, we design two algorithms, namely the dormant probing algorithm, and a variable heartbeat probing algorithm, to detect the availability of Internet connectivity via WiFi. We implement the algorithms and mechanisms we design on smartphones. We also implement the capability of simultaneously accessing WiFi and cellular networks on smartphones. We evaluate our design and implementation using an Internet availability simulator which is based on a two-state continuous time Markov chain. Our experimental results show that we are able to detect unavailability of Internet connectivity beyond the first hop when using WiFi and switch to the cellular network fairly quickly. Furthermore, with the variable heartbeat algorithm, we are able to switch back to WiFi within 60 seconds of Internet connectivity becoming available, 90% of the time, with minimal probing overhead.

For my Mom, Dad, Brother, and Friends

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

The past two years of my life can be summarized as a challenging and rewarding journey, a journey which would not have been possible without the help of a lot of people.

First of all, I express my sincere gratitude to my advisor, Dr. Sneha Kumar Kasera, for continually enthusing and motivating me to pursue research throughout my life in graduate school. Sneha gave me the freedom to pursue the topic of my interest and provided valuable research directions. His timely suggestions helped me progress whenever I was stuck in the course of my research. Apart from his technical expertise, I have always been inspired by Sneha's writing. He has helped me become a much better writer than I was before. I will forever cherish the many technical and casual discussions we have had and the set of skills he imparted to me as an advisor.

I would like to thank my Committee members, Dr. Jacobus Van der Merwe and Dr. Robert Ricci, for providing constructive comments, support, and guidance throughout my thesis. My independent study with Kobus was also a major factor in inspiring me towards research. I am also grateful to Ann Carlstrom for being a wonderful graduate advisor and aiding me with administrative requirements.

A lot of my friends have played a big part in my grad school life. I thank my labmates, Arijit Banerjee, Mojgan Khaledi, and Phil Lundrigan, for the many enlightening discussions and ideas. They have been very influential in reforming the way I think about a problem. I thank Arvind Haran and Anand Venkat for spending many hours on proof-reading my writing and giving good suggestions. I thank my roommates, Arun Baskar, Shashanka Krishnaswamy, and Anand Venkat, for continuously encouraging and backing me up. They provided me with adequate distraction and fun that helped me do my thesis enjoyably. Special thanks to Aravind Senguttuvan and Praveen Kumar Shanmugam for providing me with their laptops for running my experiments.

I am indebted to my parents Bhanu and Subramanyam and my brother Ashwin for having unwavering belief in me and giving me the required support and strength for doing this thesis. Last but not least, I thank God for blessing me with this excellent opportunity.

CHAPTER 1

INTRODUCTION

The use of smartphones for various data applications is growing at a tremendous rate. Over 700 million smartphones were sold in 2012 and the number crossed a billion in 2013 [1]. Smartphones, typically, have wireless access to the Internet by means of network interfaces including WiFi, 3G, or LTE. Cisco's Virtual Networking Index reports that the global mobile data traffic reached 1.5 exabytes per month at the end of 2013 [2].

Ubiquity, high speed, and affordability are three primary features that smartphone users expect from their Internet connection. WiFi is a desirable choice for Internet access for smartphone users since it offers high-speed connectivity either for free or at low prices. However, WiFi coverage is limited and might not be available, especially when a user is on the move. On the other hand, cellular networks, while offering wider coverage, provide lower data rates, consume higher power compared to WiFi, and are more expensive. Therefore, a cellular data user, optimally, would like to switch back to a WiFi connection as soon as one is available. Unfortunately, most existing devices determine the availability of the Internet through WiFi by examining the strength of the signal received from the WiFi access points. They do not necessarily check the connectivity beyond the first hop.

Internet connectivity can become unavailable beyond the first hop (Figure 1.1) during arbitrary times of the day. We have observed this behavior in cable provider networks (e.g., the Comcast network). Grover et al. [3] report that the median duration of Internet downtimes in developing countries is 30 minutes and the median duration between downtimes is less than a day. During these periods of unavailability, users might need to manually switch to a different WiFi access point or to a cellular network. Now, once a user switches to the cellular network, it must manually switch back to the WiFi network often to check whether the Internet connection via WiFi is available or not. As noted above, the user would like to switch back to WiFi to save smartphone power, to obtain a higher data rate, or to save money. This manual switching between WiFi and cellular networks can lead to Internet disconnection for a certain amount of time, wastage of power, and user frustration. Thus, an

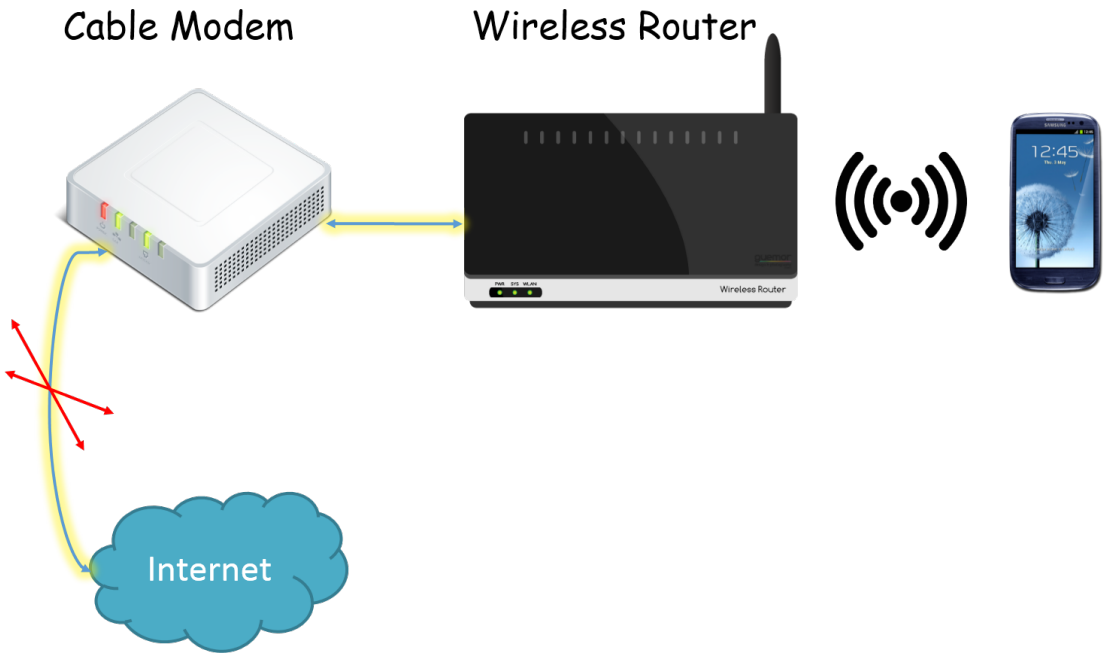

Figure 1.1: Loss of Internet connectivity beyond the cable modem

automated solution is highly desirable.

Current solutions for switching between cellular networks and WiFi, such as those provided by Google and Samsung [4], do not work well in many scenarios. As an example, a Samsung Galaxy S4 smartphone connected to a WiFi network remains in the WiFi mode for more than 5 minutes, even when the Internet connectivity beyond the first hop is unavailable. Therefore, there is a strong need for an efficient and automated switching mechanism between WiFi and cellular networks that has wide applicability. We propose an automated and efficient mechanism for switching between WiFi and cellular networks while providing robust Internet connectivity.

1.1 Thesis Statement

In this master's thesis, we design an efficient way to switch between WiFi and cellular networks, thereby providing robust Internet connectivity, while minimizing any associated overheads.

1.2 Thesis Contributions

- We design an adaptive, energy-efficient methodology to switch between WiFi and cellular networks to provide robust connectivity. When using a WiFi network, we utilize a combination of active and passive monitoring techniques to detect unavailability of Internet connectivity via WiFi. When using the cellular network, we design two

algorithms, namely the dormant probing algorithm and a variable heartbeat probing algorithm, to detect the availability of Internet connectivity via WiFi.

- We implement the algorithms and mechanisms we design on smartphones. We also implement the capability of simultaneously accessing WiFi and cellular networks on smartphones.

- We evaluate our design and implementation using an Internet availability simulator which is based on a two-state continuous time Markov chain.

1.3 Thesis Overview

The rest of the thesis is organized as follows. We describe the related work and our initial experiments to demonstrate the problems with current implementations of switching between WiFi and cellular networks in Chapter 2. In Chapter 3, we detail our design ideas on the adaptive pinging mechanism when a smartphone is connected to a cellular network and would like to switch to a WiFi network, and the passive and active monitoring techniques used when the smartphone is connected to the WiFi network and should switch to the cellular network when the Internet connectivity is not available beyond the WiFi access point. In Chapter 4, we provide our implementation details and present evaluations of our design and implementation. Chapter 5 concludes this thesis.

CHAPTER 2

RELATED WORK

The challenge of providing ubiquitous, high-speed, and affordable Internet connectivity to smartphones has gained much importance and attention in the last decade. Ananthanarayanan et al. [5] proposed to increase WiFi performance using Bluetooth signals and cell-tower information, while Ravindranath et al. [6] discuss methods to improve WiFi performance based on sensor hints.

Existing implementations are available in Android smartphones to make the Internet connectivity better. Google Android [4] has options to avoid poor WiFi connection (less signal strength) implemented with the help of a WiFi watchdog mechanism, which focuses on last-hop Internet connectivity. This work proposes that whenever the RSSI values are lower than a threshold, the smartphone gets disassociated from the access point, thereby providing connectivity to Internet via cellular networks.

Samsung smartphones have a custom services layer of Android OS. Version 4.1 is enabled with an option that checks for availability of Internet connection via WiFi and reports if it is not available. From version 4.3 onwards, it also switches to the cellular network and switches back to WiFi. However, there are lot of problems associated with such solutions in terms of correctness and efficiency (elaborated in Section 2.2.1).

Network Connectivity Status Indicator (NCSI) in Microsoft's Windows operating systems, is a network awareness module that reports when the Internet Connectivity via an interface is not available. At certain time intervals, an HTTP GET and a DNS Query are sent to preset websites [7]. Based on the response, appropriate notifications are provided to the user. We conduct experiments with NCSI to understand how it behaves and one of the observations is that there are cases where it takes as long as 3 minutes to detect unavailability of the Internet connectivity and notify the user. Complete results are reported in Section 2.3. Our efficient methodologies will detect unavailability of the Internet connectivity faster than NCSI.

2.1 Initial Experiments

We design experiments with the intent to understand the behavior of existing solutions and to evaluate their performance. Specifically, we conduct experiments with Android smartphones under six possible scenarios, and also experiment with Microsoft's NCSI.

2.2 Experimenting With Android Smartphones

2.2.1 Scenarios for experimentation

We identify the following six possible scenarios and observe the change in behavior of existing systems in response to the onset of events.

1. Initially, WiFi is on and Internet connectivity is available via a cable modem; subsequently, we disconnect the cable from the cable modem, thereby losing network connectivity beyond the first hop.

2. After 1, restore the Internet connectivity via WiFi by plugging the cable into the modem.

3. Connect to a WiFi access point with no Internet connectivity.

4. Switch to WiFi from the cellular network. Initially, WiFi is switched off in the smartphone and cellular network is used.

5. Perform 4, but Internet connectivity via WiFi is not available.

6. Perform 5 and after some delay, restore the network connectivity beyond the first hop.

2.2.2 Results of experiments on different versions of Android

To understand the behavior of different versions of Android which are being used with popular smartphones in the market, we experiment with the following smartphones: Samsung Galaxy S3, S4, Google Nexus 4. The results are shown in Table 2.1.

Some of the behaviors recorded in Table 2.1 were traced back to the relevant portion of the source code[1] that causes them. One of the main reasons for LG Nexus with Android 4.3 performing poorly is that the WiFi watchdog monitor's state changes are triggered by RSSI going below a certain threshold. All of our experiments were carried out from a location that had a good signal strength. For Samsung's smartphones, the source code is not available and the results are based only on the experiments we conduct.

---

[1]Source code of Android 4.3 is publicly available.

Table 2.1: Behavior of Android smartphones under different scenarios described in Section 2.2.1
Red: Not working as expected Green: Working as expected

Smartphone (Android version)	In WiFi, remove cable from modem (1)	After 1, plug back cable into modem (2)	Connect to WiFi AP with no Internet connectivity (3)	In cellular, switch on WiFi (4)	Perform 4, but with cable removed from modem (5)	Perform 5, after some delay, plug back cable into modem (6)
Samsung S3 (4.1.2)	Stays in WiFi, occasionally notifies user of no Internet connectivity	–	Gets Connected, notifies user of no Internet connectivity	Works as expected	Gets connected	–
Samsung S4 (4.3)	Some cases, stays in WiFi, some cases, switches to cellular network but takes more than 5 minutes	Some cases, immediately switches to WiFi, other cases does not switch	Works as expected	Works as expected	Works as expected	Does not go back to WiFi
LG Nexus (4.3)	Stays in WiFi	–	Gets connected	Works as expected	Gets connected	–
Expected Behavior	Should switch to cellular network	Should switch back to WiFi	Does not associate	Should switch to WiFi	Should not switch back to WiFi	Should switch back to WiFi

We observe that Samsung S3 notifies the user that there is no Internet connectivity, but does not switch to the cellular network. Samsung S4 has a modified version of the service layer provided by stock Android and hence, it performs better than the others but fails under two conditions and does not perform strongly in one case. As an example, when Internet connectivity via WiFi is not available (case 1), in some instances, S4 takes more than 5 minutes to switch to the cellular network and in other instances, it just stays in WiFi. When it switches to the cellular network, it adds the WiFi access point into a poor connection list and notifies the user that it avoided a poor Internet connection. In case 5, when we turn on WiFi without having Internet connectivity beyond the first hop while being in the cellular network, it checks for the Internet connection and since it is not available, the smartphone does not associate with the access point and adds the WiFi access point into a poor connection list. At this moment, when we enable connectivity beyond the first hop in WiFi, the smartphone stays in the cellular network (case 6).

2.3 Experimenting With Microsoft NCSI

Microsoft's Network Connectivity Status Indicator (NCSI) [7] is a part of the network awareness module in the Windows operating systems. Network Awareness collects network connectivity information and makes it available through an application programming interface (API) to services and applications on a computer. NCSI can identify if there is connectivity to the Internet, including the ability to send a DNS request and get it resolved. NCSI is designed to be responsive to network conditions, hence it examines the connectivity of a network in a variety of ways. For example, NCSI tests connectivity by trying to connect to http://www.msftncsi.com, a minimal web site that exists only to support the functionality of NCSI.

To understand how NCSI performs under changing Internet conditions, we perform experiments with both wired and wireless network connections and the results tend to be the same. Wireshark traces of packets are captured for over 60 minutes in a laptop with Windows operating system. Here are some observations based on wireshark traces:

- While connecting to a wired network or a wireless access point, an HTTP GET and a DNS request is sent and the user is notified when there is no reply.
- For the first couple of minutes, there seem to be periodic requests being sent to verify Internet connectivity, following which they cease.
- After Internet is disconnected, there are cases where it takes as long as 3 minutes to detect unavailability of the Internet connection.

So, the existing solutions do not work properly and we design solutions to detect the unavailability of Internet connectivity via WiFi (while using WiFi) and availability of Internet connectivity via WiFi (while using cellular network) faster than the existing solutions.

CHAPTER 3

DESIGN

In this chapter, we present our design methodology.[1] Our primary design goals are efficiency (sending less probes, utilize less resources), robustness (smartphone connected to the Internet whenever it is available), and minimizing the energy cost (minimizing the amount of time spent in cellular networks).

3.1 Overview

Our design includes an adaptive probing mechanism utilizing a combination of passive monitoring and active probing techniques for checking Internet connectivity and switching between WiFi and cellular networks as required.

Initially, a smartphone can either be using a WiFi or a cellular network. While using WiFi, we passively monitor the Internet connectivity periodically (say, every $n1$ seconds) and if there is a necessity to do a ping, it will be sent to check the availability of Internet connectivity (more in Section 3.2). If the Internet connectivity is not available, the smartphone is switched to the cellular network, otherwise, it goes to sleep and continues passive monitoring. While using the cellular network, Internet connectivity via WiFi is checked periodically (say, every $n2$ seconds) without disrupting the connectivity via cellular network. If the Internet connectivity via WiFi is available, the smartphone is switched back to WiFi, otherwise, it goes to sleep mode and tries again after $n2$ seconds. The values of $n1$ and $n2$ should be different because the duration of time Internet connectivity via WiFi is not available is different from the duration of time it is available. Also, the value of $n2$ will change based on the nature of connectivity (more in Section 3.3).

We start by explaining our probing technique for detecting Internet disconnectivity while being connected to the WiFi network, followed by our adaptive probing technique for detecting Internet availability via WiFi when the cellular network is being used.

---

[1] We have used Google's Android OS in our design, methodologies, and examples

3.2 Probing While Using WiFi Network

To detect unavailability of Internet connectivity via WiFi, a reliable server must be pinged. Time interval between the pings is an important parameter of our design. Pings can be sent at a periodic interval or sent adaptively based on the quality of the connection.

We propose a solution that uses a combination of passive monitoring and active probing techniques. We start with the passive monitoring technique by monitoring the network statistics and if there is some anomalous behavior observed with the statistics, we send a ping to the reliable server for checking Internet connectivity.

The process is explained in Algorithm 1. Passive monitoring is done by checking the network statistics. Network-related statistics are written into a file */proc/net/snmp* in the Android file system. This file has counters for outgoing, incoming, and retransmitted TCP segments and outgoing and incoming UDP datagrams apart from many other counters. Whenever the Internet connectivity is not available and one of the applications in the smartphone tries to use the network, the number of outgoing segments increases but not the incoming segments. After a certain interval, even the retransmitted segments increase as the application tries to retransmit the packets. This increase in the outgoing and retransmitted segments and no change in the incoming segments is an anomalous condition, indicating that the Internet connectivity might not be available. To verify if the Internet connectivity is available or not, we ping a reliable server.

If there is no change in incoming and outgoing segments, we go to sleep mode and passively monitor again after $n1$ seconds. If this continues for r consecutive intervals, a ping will be sent to a reliable server. This is because there might be push notifications,[2] which might be missed if the Internet connectivity is not available and WiFi network is being used. If the Internet connectivity is not available, we switch to the cellular network as explained in Section 3.4.1. We describe the selection of values for the parameters $n1$ and r in Section 4.2.5.1.

3.3 Probing WiFi While Using Cellular Network

When Internet connectivity via WiFi is not available, we switch to the cellular network if it is available. If there is no cellular connection, the switch will not be done.

Cellular data are expensive and also drain the battery quicker than being in WiFi. Therefore, as and when Internet connectivity via WiFi is available, we should switch back to WiFi. Also, the user must not see any disruption while we test the availability of Internet connectivity via WiFi.

---

[2]Cloud services can send notification messages directly to apps on mobile devices. These messages are called push notifications.

Algorithm 1: Detecting unavailability of Internet connectivity while in WiFi

```
1  initialization;
2  InternetAvailable=True;
3  while InternetAvailable && connectedToWiFi do
4      read network counters;
5      if (increase in outgoingCounter or retransmittedCounter) && no change in
       incomingCounter then
6          ping google.com;
7          if ping not succeeded then
8              InternetAvailable = False;
9      else if no change in counters 'r' consecutive times then
10         ping google.com;
11         if ping not succeeded then
12             InternetAvailable = False;
13     else
14         change in counters, traffic flowing normally, don't do anything
15     sleep 'n1' seconds;
16 switch to cellular network
```

One advantage with Android smartphones is that both WiFi and cellular network can be used at the same time (though this is not the usual case and seldom used). We will exploit this advantage and test availability of Internet connectivity via WiFi, even while the cellular network is used by applications on the smartphone, so that the user does not notice anything unusual. This process of testing Internet connectivity via WiFi while being in the cellular network is explained in Section 3.4.2.1. If there is no WiFi connectivity available, the smartphone will not be connected to any WiFi access point and we will not be doing anything until the WiFi connection is available.

We propose two different adaptive probing approaches for testing Internet connectivity via WiFi, namely *dormant adaptive probing* and *variable heartbeat adaptive probing*. In the dormant adaptive algorithm, we sleep for a long time before checking the availability of Internet connectivity via WiFi (hence the name dormant), while in the variable heartbeat algorithm, we check for availability of connectivity from time to time.

3.3.1 Dormant adaptive probing algorithm

Algorithm 2 explains this approach. The first time we switch into the cellular network, we ping via the WiFi network every t seconds (frequent probing), and obtain the disconnection time (estimated time) x_{old} seconds, when the Internet connectivity via WiFi is available (see Figure 3.1). For subsequent switches, we sleep for estimatedTime seconds and then probe. On success, x (sample time) is the new disconnection time (Figure 3.2), and on failure, sleep

Algorithm 2: Dormant adaptive probing algorithm while in cellular network

```
1  initialization;
2  InternetAvailable = False
3  if switchingForFirstTime then
4  |   switchingForFirstTime = False
5  |   probe every t seconds, x_old = disconnectionTime
6  else
7  |   sleep x_old and probe
8  |   if probe success then
9  |   |   x = x_old
10 |   else
11 |   |   timeInCellular = x_old
12 |   |   while !InternetAvailable do
13 |   |   |   sleep t seconds
14 |   |   |   timeInCellular += t
15 |   |   |   if probe success then
16 |   |   |   |   x = timeInCellular
17 |   |   |   |   InternetAvailable = True
18 |   if timeInCellular < x_old then
19 |   |   reduce t by r seconds
20 |   //calculate EWMA (exponential weighted moving average)
21 |   x_old = (1 − α) ∗ x + α ∗ x_old
22 |   switch to WiFi
```

another t seconds and probe again (Figure 3.3).

If the Internet connectivity via WiFi is available after the estimated time (x_{old}) for s consecutive switches, reduce sampleTime (x) by r seconds. Then, calculate the exponential weighted moving average (EWMA) x_{old} (estimatedTime) of x and past history using this equation,

$$estimatedTime = (1 - \alpha) * sampleTime + \alpha * estimatedTime$$

The value of α is chosen in such a way that the extreme values does not affect the estimated time. With this dormant algorithm, it might take a very long time to detect the availability of Internet connectivity via WiFi. This is because, at certain instances, the estimated disconnection time can be very high and the algorithm will be in the sleep mode for a long time. So we came up with another version of this algorithm called the variable heartbeat adaptive probing algorithm as explained below.

3.3.2 Variable heartbeat adaptive probing algorithm

One of the problems with the dormant algorithm is that the algorithm sleeps for the entire estimated time before a probe is sent to check the Internet connectivity via WiFi. But during

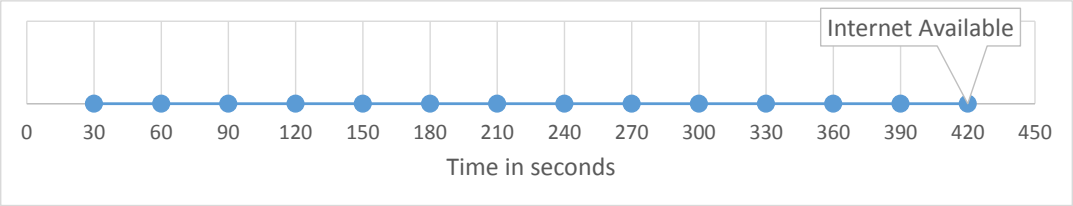

Figure 3.1: Frequent probing in dormant adaptive algorithm: On the first switch into the cellular network, we probe frequently. Here, every blue dot represents a probe and interval between probes is 30 seconds. Internet connectivity via WiFi is available at 420 seconds and that will be the estimatedTime.

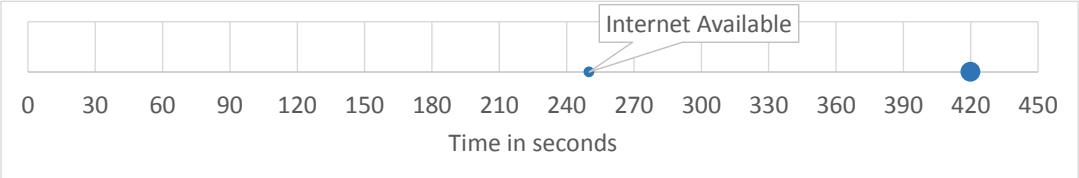

Figure 3.2: Probing in dormant adaptive algorithm: Estimated time from Figure 3.1 is 420 seconds and hence, we sleep for 420 seconds before checking for Internet connectivity via WiFi. Internet connectivity via WiFi is available at 250 seconds and since we detect at 420 seconds, the sample time is 420 seconds.

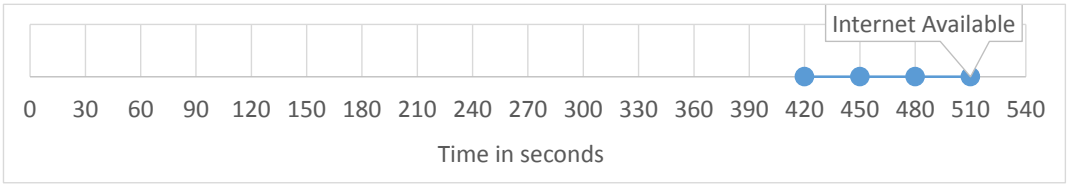

Figure 3.3: Probing in dormant adaptive algorithm: After sleeping for the estimated time (420 seconds), if Internet connectivity via WiFi is not available, we resort to frequent probing and probe every 30 seconds in this example. We detect the availability of Internet connectivity via WiFi at 510 seconds and this is the sample time.

the sleep time, if the Internet connectivity via WiFi is available, we fail to detect it.

To overcome this problem, we need an algorithm which starts by sleeping t seconds and increases multiplicatively till the estimated disconnection time or a particular threshold. To put it formally, everything is similar to the dormant algorithm, but instead of sleeping for the entire estimated time (x_{old}), we sleep for t seconds, increase multiplicatively, on reaching a threshold, increment by t seconds and try again. For example, if estimatedTime is 420 seconds and t is 30 seconds, the algorithm will probe periodically at t = 30, 90, 210, 240, 270, 300, ... etc. sleeping between the probes (Figure 3.4). Here, the threshold is half of estimatedTime and once sleep time reaches 210, we just sleep for 30 seconds every time, but do not increase multiplicatively because we want to switch back to WiFi as soon as Internet connectivity is available. Algorithm 3 explains this approach. We describe the selection of values for the parameter t in Section 4.2.5.2.

3.4 Switching Between WiFi and Cellular Networks

There are some design problems associated with switching between WiFi and cellular networks. In this section, we explain them and provide solutions. The two main problems are:

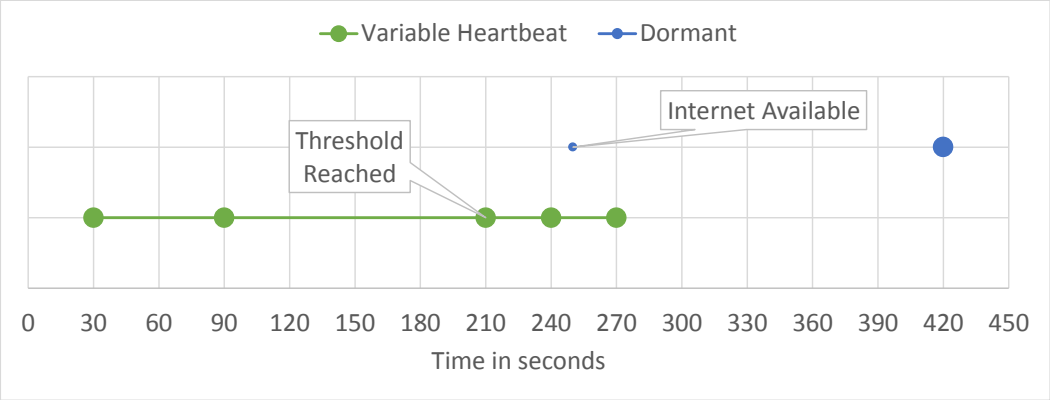

Figure 3.4: Variable heartbeat adaptive algorithm - comparison with dormant probing algorithm: Here, the estimatedTime is 420 seconds. While the dormant probing algorithm sleeps for the entire 420 seconds and then probes, the variable heartbeat algorithm probes at 30, 90, 210 seconds. Threshold is half of estimatedTime and is set as 210. Once the threshold is reached, we probe frequently till Internet connectivity via WiFi is available. In this example, Internet connectivity via WiFi is available at 250 seconds, and the variable heartbeat algorithm detects at 270 seconds, while the dormant probing algorithm detects at 420 seconds.

Algorithm 3: Variable heartbeat adaptive probing algorithm while in cellular network

```
initialization;
InternetAvailable = False
if switchingForFirstTime then
    switchingForFirstTime = False
    probe every t seconds, x_old = disconnectionTime
else
    sleep for t seconds and probe
    while !InternetAvailable do
        timeInCellular += t
        if probe success then
            x = timeInCellular
            InternetAvailable = True
        else
            if t < x_old/2 then
                t *= 2
            else
                //use initial value of t
            sleep for t seconds
    if timeInCellular < x_old then
        reduce t by r seconds
    //calculate EWMA (exponential weighted moving average)
    x_old = (1 − α) ∗ x + α ∗ x_old
    switch to WiFi
```

1. Switching to the cellular network from the WiFi network.

2. While being connected to the cellular network, determining whether WiFi has recovered or not and determining it without disconnection.

Solutions to these problems are described below:

3.4.1 Switching to the cellular network

After identifying that the Internet connectivity via WiFi is not available, we need to switch to a different network if it is available. Usually in Android-based smartphones, cellular data can never be switched on by a user if WiFi is on and if it is connected to an access point. Hence, in order to switch to cellular data, we turn off WiFi and turn on cellular data, so that the smartphone uses the cellular network.

3.4.2 Determining availability of Internet connectivity via WiFi without disconnecting from the cellular network

While in the cellular network, to determine whether Internet connectivity via WiFi is available or not, we need to have both WiFi and cellular network interfaces on at the same time. This is to ensure that the smartphone is not disconnected from the cellular network when the availability of Internet connectivity via WiFi is determined.

3.4.2.1 Accessing both WiFi and cellular data at the same time

Officially, simultaneous access of WiFi and cellular data is not supported by Google Android. There are three existing approaches to perform this simultaneous access. First, [8] explains an approach which works only on a rooted smartphone. Second, [9] suggests an approach of hacking the Android source code, and third is an approach by which Java's reflection can be used to modify the behavior in run time [10].

The problem with the second approach is that the source code must be changed by removing the condition which enables only one interface to be active at a time. Another problem is that, since the changes are done in the source code, it must be compiled and installed on the smartphone and also with new releases, code must be changed, compiled, and installed. The problem with the third approach is that it is not supported by most vendors and may break anytime.

Hence, out of these three approaches, the first [8] approach is the most feasible one, so this is used to make WiFi and cellular data work at the same time. Note that there is no API support for this approach.

Android OS removes the WiFi kernel module once WiFi is switched off. So in order to test the availability of Internet connectivity via WiFi, every step that is performed by Android when WiFi is turned on should be performed, i.e., loading the kernel module, bringing up the interface, associating the smartphone with an access point, getting an IP address using DHCP, etc. Once these steps are performed, WiFi will be turned on (more details in Section 4.1.1). While all the applications on the smartphone use the cellular network, only our application uses WiFi to test the availability of Internet connectivity. We perform these steps every time we want to check for Internet connectivity via WiFi and once we perform the check, we revert the steps to remove the WiFi module and if the Internet connectivity via WiFi is available, we switch on WiFi through the API.

CHAPTER 4

IMPLEMENTATION AND EVALUATION

In this chapter, we discuss our implementation, the evaluation methodology, and present the results of our evaluation. We start by giving the implementation details, followed by the questions which we answer through this evaluation, experimental setup, the metrics, factors varied, and the results.

4.1 Implementation

We implement the algorithms and the switching mechanism described in Chapter 3 in HTC Nexus one smartphones powered by the CyanogenMod[1] version 7 operating system. Busybox[2] tools, which configures various utilities like iwconfig, insmod etc., is installed to be used in the smartphone.

We write an Android application that starts a background service which realizes the passive and active monitoring techniques described and switches between the WiFi and cellular networks appropriately. The Android SDK (Software Development Kit) provides APIs for common wireless networking functions like switching the WiFi interface on and off, but does not provide APIs for keeping both the WiFi and cellular interfaces on at the same time. We spawn a separate process from the application which enables the WiFi interface when the cellular network is active, as described below in Section 4.1.1.

---

[1] CyanogenMod is a customized, aftermarket firmware distribution for several Android smartphones, which is designed to increase performance and reliability over Android-based ROMs released by vendors and carriers such as Google, T-Mobile, HTC, etc. CyanogenMod also offers a variety of features and enhancements that are not currently found in these versions of Android [11].

[2] BusyBox is a software application that provides many standard Unix tools, much like the larger GNU Core Utilities. BusyBox is designed to be a small executable for use with the Linux kernel, which makes it ideal for use with embedded devices. It has been self-dubbed "The Swiss Army Knife of Embedded Linux." Some of the utilities used here are iwconfig, insmod, and rmmod [12].

4.1.1 Concurrent WiFi and cellular access

As mentioned in Section 3.4.2.1, both the WiFi and cellular interfaces can be accessed at the same time. Below, we list the steps we do to use both the interfaces simultaneously.

1. Switch off WiFi in the smartphone and switch on cellular data. Once cellular data use is initiated, we need to enable the WiFi interface (eth0).

2. Load the WiFi driver module into the smartphone. The driver is platform-dependent and is usually located in the directory */system/lib/modules*.

   ```
   busybox insmod /system/lib/modules/bcm4329.ko firmware_path = /system/ve
   ndor/firmware/fw_bcm4329_apsta.bin
   ```

3. Use wpa_supplicant[3] to bring up the WiFi interface. This will connect the smartphone to the access point.

   ```
   wpa_supplicant -B -Dwext -ieth0 -c/data/misc/wifi/wpa_supplicant.conf
   ```

   -B option is used to run the daemon in the background and wext is the generic wireless extensions driver for Linux.

4. Use the dhcpd daemon to get an IP address from the access point. This also installs default routes via WiFi.

   ```
   dhcpcd eth0
   ```

5. Now both the interfaces are enabled and the smartphone will have two default routes (one via WiFi and another via the cellular network). We delete the default route via WiFi so that the cellular network will be used by applications on the smartphone while we test the Internet connectivity via WiFi.

   ```
   ip route del default via 192.168.2.1 dev eth0
   ip route add <reliable-server-ip-address> via 192.168.2.1 dev eth0
   ```

   Here, 192.168.2.1 is the first hop IP address from the smartphone.

---

[3]wpa_supplicant is an implementation of IEEE 802.11i supplicant. wpa_supplicant is designed to be a "daemon" program that runs in the background and acts as the backend component controlling the wireless connection.

4.2 Evaluation and Results

4.2.1 Questions answered by this evaluation

These are the questions we answer through this evaluation.

1. How well will the system detect disconnections of Internet via WiFi?
2. Which of the variable heartbeat or dormant adaptive algorithms detects the availability of Internet connectivity via WiFi faster?
3. How many probes are sent?

4.2.2 Experimental setup

Figure 4.1 shows our experimental setup. We configured a laptop running Ubuntu 13.04 to create a wireless hotspot. The laptop is connected to the Internet via the Ethernet and the wireless card is used to create the WiFi hotspot. In Ubuntu, hotspots are run with the help of iptables[4] forwarding rules. We will be running the Internet availability simulator in this laptop. The Internet availability simulator will remove forwarding rules and adds rules to drop the packets when it transitions into the Off state and, when transitioning into the On state, the forwarding rules are added and the packet drop rules are deleted. Smartphones running our code are connected to the Internet through this WiFi hotspot.

The Internet availability simulator logs every transition into the On and Off states; our Android application also logs all the switches between WiFi and cellular networks. For one of the metrics, the time taken to detect the availability of Internet connectivity via WiFi by our algorithm, we find the difference between the time when the algorithm switches to the WiFi network from the cellular network and the time when the Internet availability simulator transitions into the On state. For accurate measurements, time is synchronized[5] with pool.ntp.org before the experiments in both the smartphones and in the laptop running Ubuntu. Smartphones also run the traffic generator application to generate periodic traffic.

Since passive monitoring is used in our algorithm, ping will not be sent until we observe anomalous stats. We need an application on the smartphone which generates traffic, so that anomalous stats will be detected when Internet connectivity is not available. We have written

---

[4]Iptables is used to set up, maintain, and inspect the tables of IPv4 packet filter rules in the Linux kernel. Several different tables may be defined. Each table contains a number of built-in chains and may also contain user-defined chains.

[5]We use 'Smart time sync' application on Android and ntpdate command on Ubuntu to synchronize the time.

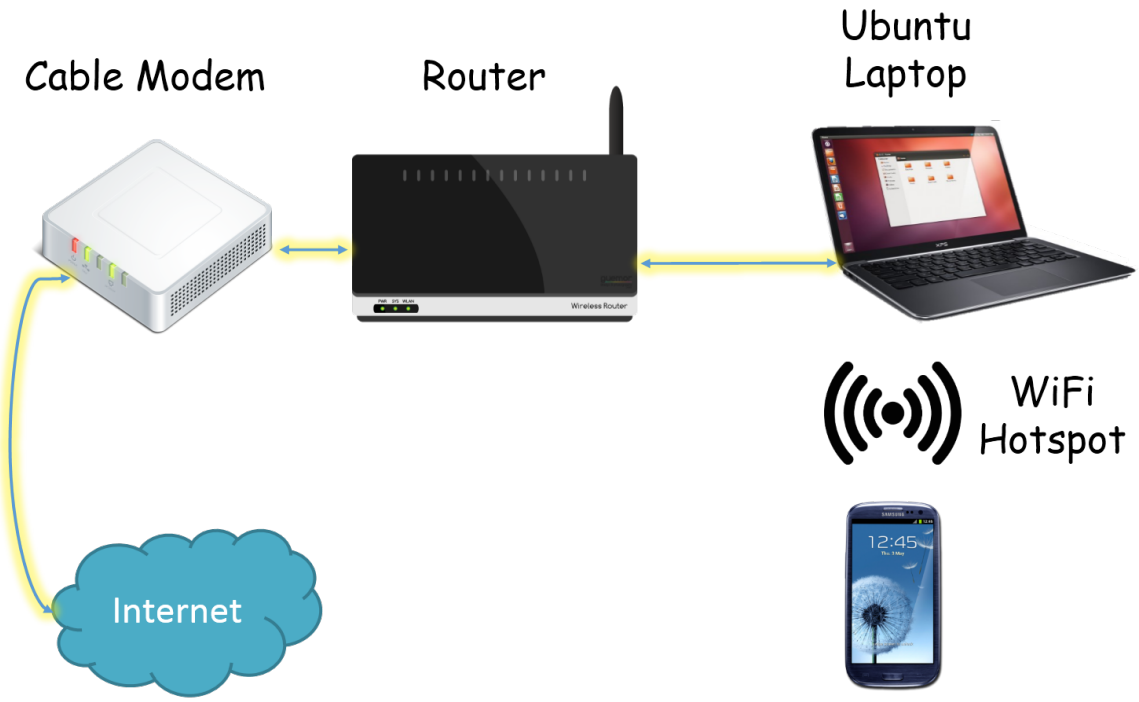

Figure 4.1: Experimental setup

an Android application that requests a web page every 60 seconds [13] and this application is run during the experiments.

4.2.3 Internet availability simulator - On-Off model

In order to evaluate the algorithms, we need data on Internet availability. Measuring Broadband America [14] is a program to monitor the cable broadband connections, conducted by the FCC[6]. From this program, we obtained the data on Internet availability. On analyzing them, we observed that there were lots of false positives (Internet not being available for a large duration of time) and this was confirmed through email exchanges with the people behind this program.

Hence, to evaluate our system, we use an Internet availability simulator. In this section, we explain the fault model for the Internet we have designed based on the two-state continuous time Markov chain $\{X_t\}$ where $X_t \in \{0, 1\}$ ([15], [16]). Whenever Internet connectivity is available, the simulator is in the On state (state 0) and if not, the simulator is in the Off state (state 1). The simulator stays in the On state with a particular probability $p1$ and transitions to the Off state with a probability $1 - p1$. The simulator stays in the Off state with a probability of $p2$ and transitions to the On state with probability $(1 - p2)$. We model

[6]FCC is Federal Communications Commission.

this as a Poisson process, where the time interval between the decisions on whether or not to perform state transitions is exponentially distributed. We use this two-state Markov chain because it provides the capability to model bursts of Internet unavailability while the other models do not. We can define how much time the simulator stays in the Off state. These probabilities $p1$ and $p2$ are calculated based on the given time interval between the decisions, total unavailability percentage, and the bursts of Internet unavailability. Figure 4.2 shows the simulator we design.

To calculate these probabilities $p1$ and $p2$, we need to know the transition rates between the states. The transition rates of Markov Chain $\{X_t\}$ is given by this infinitesimal generator matrix Q,

$$Q = \begin{bmatrix} -\mu_0 & \mu_0 \\ \mu_1 & -\mu_1 \end{bmatrix}$$

where, μ_0 is the transition rate from state 0 to 1 and μ_1 is the transition rate from state 1 to 0. The stationary distribution associated with this Markov chain are $\pi = (\pi_0, \pi_1)$, where $\pi_0 = \mu_1/(\mu_0 + \mu_1)$ and $\pi_1 = \mu_0/(\mu_0 + \mu_1)$. Let $p_{i,j}(t)$ denote the probability that the process is in state j at time $t+\tau$, given that it was at state i at time τ, $p_{i,j}(t) = P(X_{t+\tau} = j | X_\tau = i)$. It is given by,

$$p_{i,j}(t) = \begin{cases} \mu_1(1 - exp(-(\mu_0 + \mu_1))t)/(\mu_0 + \mu_1) & i = 1, j = 0 \\ \mu_0(1 - exp(-(\mu_0 + \mu_1))t)/(\mu_0 + \mu_1) & i = 0, j = 1 \\ (\mu_0 + \mu_1(1 - exp(-(\mu_0 + \mu_1))t))/(\mu_0 + \mu_1) & i = 1, j = 1 \\ (\mu_1 + \mu_0(1 - exp(-(\mu_0 + \mu_1))t))/(\mu_0 + \mu_1) & i = 0, j = 1 \end{cases} \tag{4.1}$$

The inputs to this model are the Internet unavailability probability p, bursts of Internet unavailability b, and the time interval between the decisions λ. Bursts of Internet unavailability b is given by this equation,

$$b = E(X) = \sum_{i=1}^{\infty} 1 - P(X \leq i) = \pi_1 \frac{1}{1 - p_{1,1}(\Delta)} \tag{4.2}$$

With these, the transition rates μ_0 and μ_1 are given by the following equations,

$$\mu_0 = -p\lambda \ln(1 - \frac{1}{b}) \tag{4.3}$$

$$\mu_1 = \mu_0 \frac{1-p}{p} \tag{4.4}$$

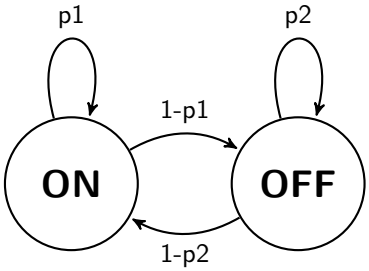

Figure 4.2: Two-state continuous time Markov chain - On-Off model

We use Python to implement the simulator, it takes p, λ, and b as input parameters and $p_{i,j}$ is calculated at time t, which is the time interval between two decisions which is modeled to be exponentially distributed.

In order to verify the correctness of the model, we run tests with various Internet unavailability and bursts of unavailability and verify that the resulting values match with the expected values. We choose the time interval between the decisions to be really low in the order of 5ms ($\lambda = 200$). This low value for interval will help us determine the correctness of the system faster than the higher values.

4.2.4 Metrics and factors used

4.2.4.1 Metrics

These are the metrics we use for this evaluation.

- Time to detect the availability of Internet connectivity via WiFi (when in the cellular network).
- Number of probes sent (overhead due of probing).

4.2.4.2 Factors varied

- Internet unavailability time: 1%, 10%, 20%, and 30%.
- Bursts of Internet unavailability: 20x and 40x the time between decisions.
- Time interval between decisions: 30, 60, and 120 seconds.

Typically, Internet unavailability time and bursts of unavailability are smaller than this. Since it is practically infeasible to evaluate our algorithms in a reasonable amount of time with such smaller unavailability time, we choose a larger value.

4.2.5 Selection of values of parameters in the algorithm

In this section, we explain how we determine the different parameters in our algorithm to be used in the experiments.

4.2.5.1 Selection of n and r for probing while using WiFi network

While using the WiFi network, we use a combination of active and passive monitoring techniques to detect unavailability of Internet connectivity via WiFi. Passive monitoring is performed every n seconds and if there is no change in the network statistics for r consecutive intervals, we send a ping to the reliable server. These values n and r will be tunable parameters in our design/implementation. Qian et al. [13] reports that the frequency of push messages from the smartphone applications is dominated by a value of 60 seconds. Hence, we use 30 seconds as n and r as 2, i.e, we passively monitor the network counters every 30 seconds and if there is no change in statistics for 2 consecutive intervals, we send a ping to the reliable server. However, as the characteristics of the applications change, the parameters can be changed.

4.2.5.2 Selection of t for probing WiFi while using cellular network

While using the cellular network, we probe the WiFi network and the dormant and the variable heartbeat algorithms use a parameter t, which will be varied according to the connectivity of the network.

Qian et al. [13] reports that the periodic transfers in mobile data are prevalent in today's smartphone traffic and that the value of 60 seconds dominates the periodicity, which is used by many of the popular smartphone applications (like Facebook, Pandora etc.). These periodic transfers are caused by multiple factors such as polling (even though cloud-based push services are available, there are applications out in the market which use polling and some poll every 20 seconds), keep-alive messages for push-based services, advertisement transfers (some are refreshed every 15 seconds), and user-behavior measurement.

There would be a periodic transfer every 60 seconds and this will lead to the cellular network interface going into a high power mode for few seconds during the transfer of data, and then, it goes through the tail phase (transition from high to low power mode), which is around 11.5 seconds for LTE. So switching to WiFi as soon as the Internet connectivity via WiFi is available is good for performance.

We experiment with two values (30, 45 seconds) of t with our variable heartbeat algorithm under two different unavailability times (10% and 30% Internet unavailability time) and measured the detection time of Internet availability via WiFi. Figure 4.3 shows the CDF of

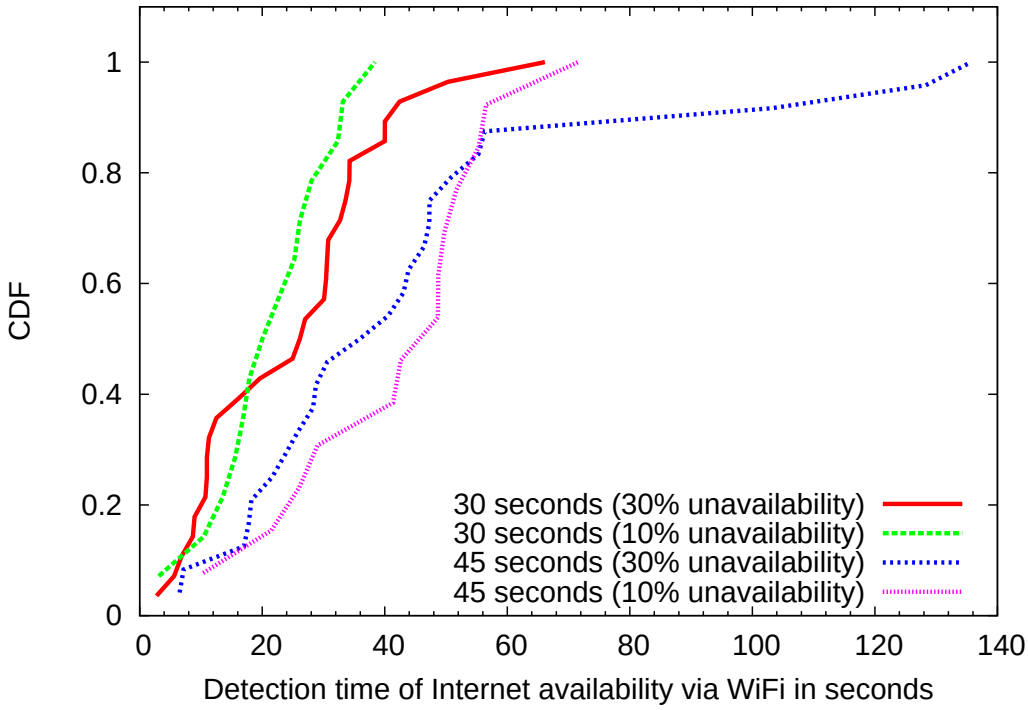

Figure 4.3: Experiment with different values of t with the variable heartbeat algorithm

this experiment. As we can see, with t as 30, the algorithm performs well with both 10% and 30% unavailability, and with t as 45 seconds, detection of Internet availability via WiFi takes a long time. Also, the Semi-Interquartile range for the values with 45 seconds is higher than 30 seconds. Hence, we choose 30 seconds as the value of t for our evaluation.

4.2.5.3 Selection of value for α to be used in EWMA

We choose the value of α to be used in the calculation of the exponential weighted moving average in such a way that extreme values do not affect the estimated disconnection time. We start with the value of 0.875 used by TCP in the congestion control algorithm and also experiment with different values like 0.8 and 0.95 and observe that these values of α do not affect the results. Hence, we use 0.875 as the value of α.

4.2.6 Experiments and results

We run the experiments with 10%, 20% and 30% unavailability time and bursts of Internet unavailability as 20X and 40X the time between decisions. Two smartphones are used, one

running the variable heartbeat algorithm and the other running the dormant algorithm. We used a time interval between decisions as 60 seconds[7] in the Internet availability simulator. Each experiment was run for 6 hours so that the Internet availability simulator could make enough transitions (at least 10) between On and Off states. These are the results of the experiments.

4.2.6.1 Detection of unavailability of Internet connectivity

Figures 4.4 and 4.5 show all the transitions the smartphone has performed between WiFi and cellular networks as the simulator moved between the states On and Off in one of our experiments (30% unavailability and bursts of Internet unavailability as 20X the time between decisions). We detect almost all the instances when Internet connectivity via WiFi is not available but a few times there was no traffic in and out of the smartphone and passive monitoring did not observe anomalous statistics. The algorithm switches into the cellular network sometimes when Internet connectivity via WiFi is available; this is because of DHCP timeout, an IP address is not being assigned (we retry once) and ping fails.

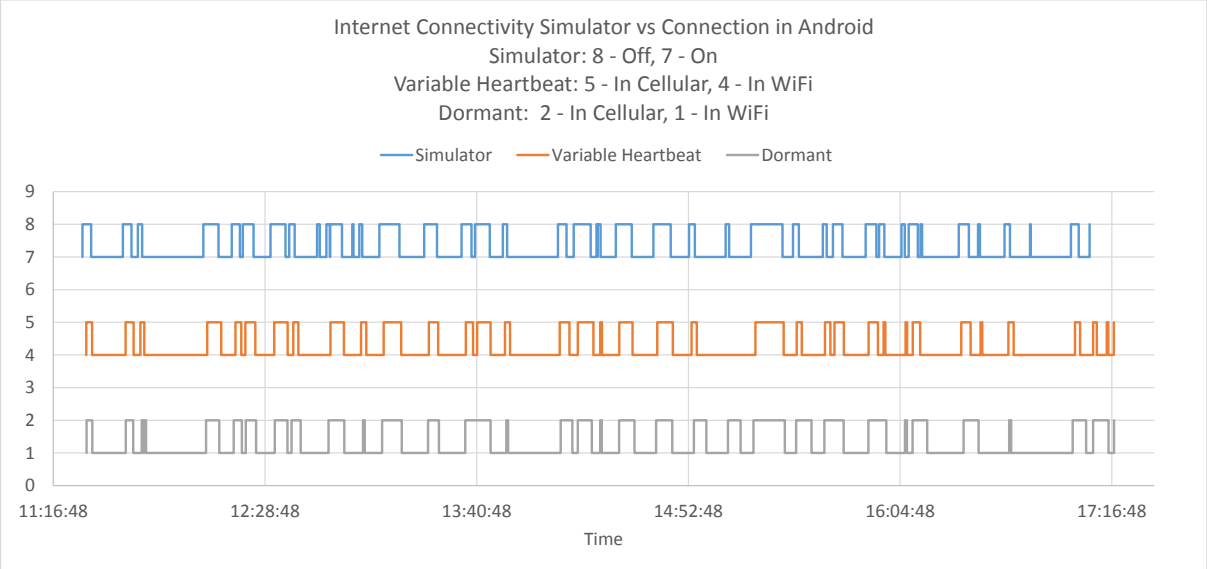

Figure 4.4: Internet connectivity simulator vs connection in Android - 30% unavailability and bursts of Internet unavailability as 20X the time between decisions

[7]We experiment with different values for time interval between decisions and observed that, with time intervals less than 60 seconds (say 10 or 30), the duration of time the simulator stays in On and Off at many instances is very small and these quick transitions between On and Off states will not be useful in evaluating our algorithm.

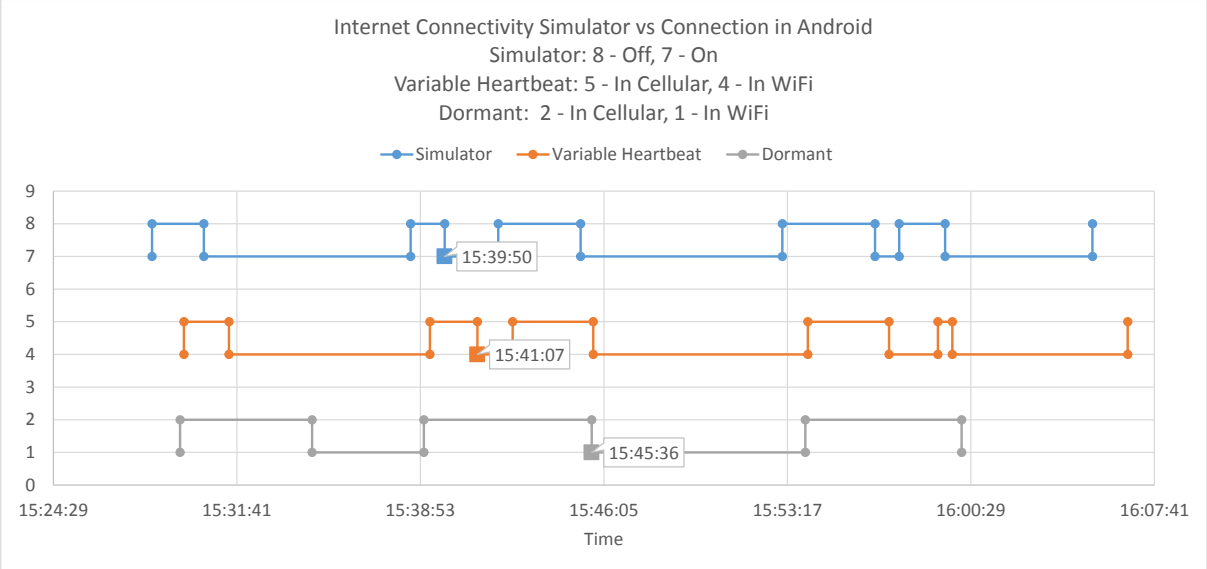

Figure 4.5: Internet connectivity simulator vs connection in Android - Sample 45 minutes zoomed in from Figure 4.4 to show the good performance of variable heartbeat over dormant algorithm.

4.2.6.2 Time to detect the availability of Internet connectivity via WiFi

Figures 4.6, 4.7, and 4.8, show the Cumulative Distribution Function (CDF) of the detection time of Internet availability via WiFi (The graphs are of different scales). As seen in the graph, the variable heartbeat adaptive algorithm performs much better than the dormant algorithm. With 10% unavailability, the variable heartbeat algorithm is able to detect the Internet availability via WiFi in less than 40 seconds (90% of the time) and, with 20% and 30% unavailability, detection happens in less than 75 seconds (90% of the time). Figure 4.9 shows the CDF of the detection times from all the unavailability results combined. 90% of the time, we detect Internet availability via WiFi within 60 seconds. We use 30 and 120 seconds as the time interval between decisions and the results are shown in Figures 4.10 and 4.11. Here too, the variable heartbeat algorithm detects the availability of Internet connectivity via WiFi faster than the dormant algorithm and the existing solutions. We also evaluate the algorithms with an ad-hoc Internet availability simulator and obtain similar results.

Poor performance of the dormant algorithm is attributed to the longer sleep times with longer downtime of Internet connection. Figure 4.5 shows the difference between variable heartbeat and dormant algorithms. As we can see from the figure, the Internet connectivity

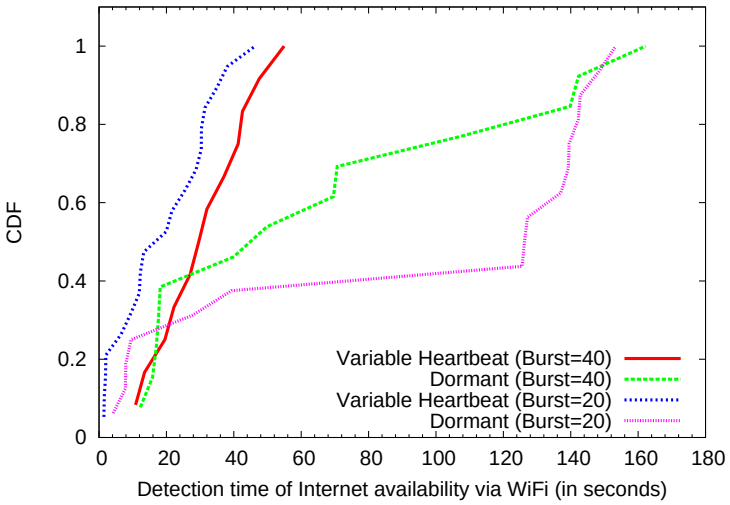

Figure 4.6: Detection time of Internet availability via WiFi (in seconds) for 10% unavailability and bursts of Internet unavailability as 20X and 40X the time between decisions

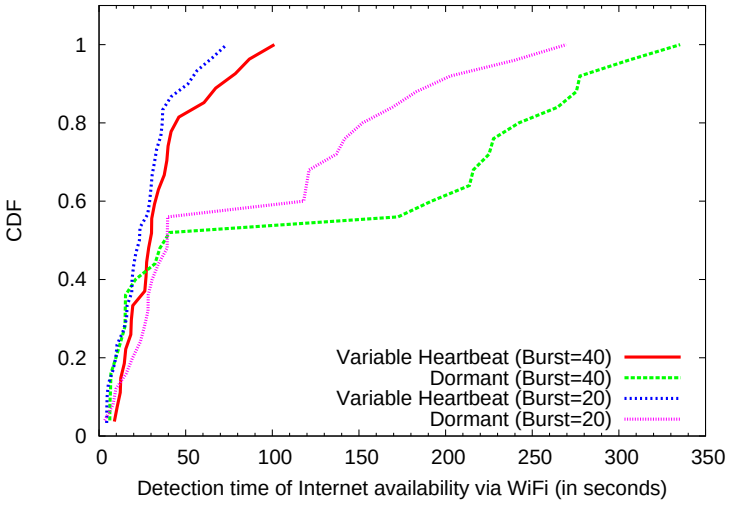

Figure 4.7: Detection time of Internet availability via WiFi (in seconds) for 20% unavailability and bursts of Internet unavailability as 20X and 40X the time between decisions

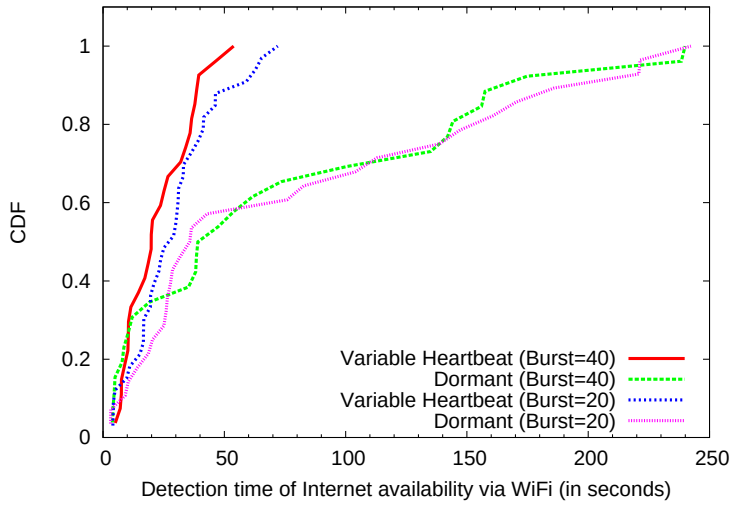

Figure 4.8: Detection time of Internet availability via WiFi (in seconds) for 30% unavailability and bursts of Internet unavailability as 20X and 40X the time between decisions

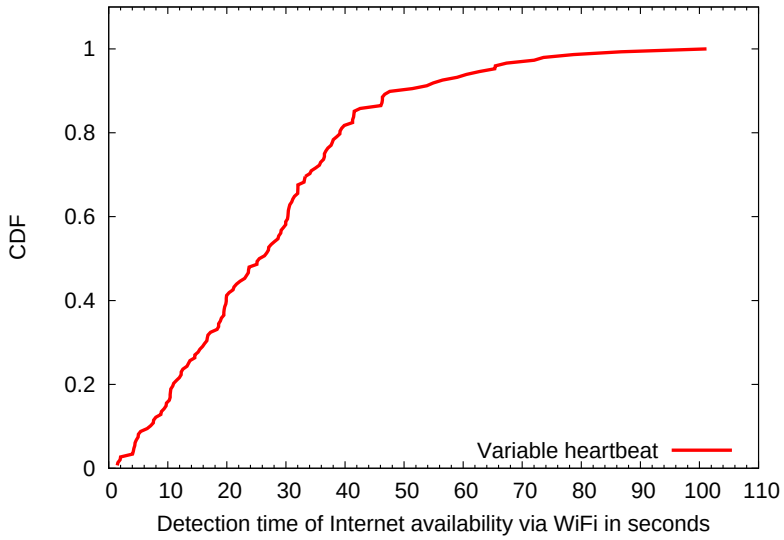

Figure 4.9: Detection time of Internet availability via WiFi (in seconds) - Variable heartbeat algorithm with the combined detection times from all the experiments.

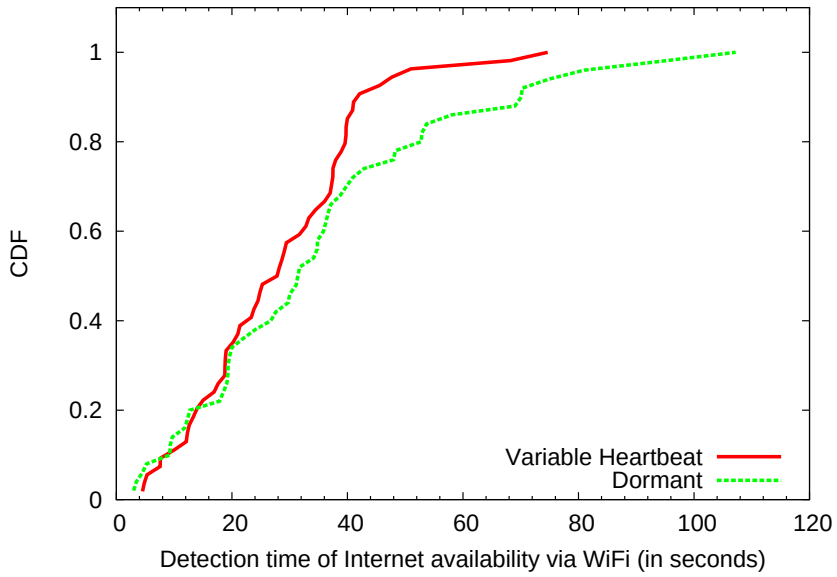

Figure 4.10: Detection time of Internet availability via WiFi (in seconds) for 30 percent unavailability, time interval between the decisions as 30 seconds, and the bursts of Internet unavailability as 40X the time between decisions

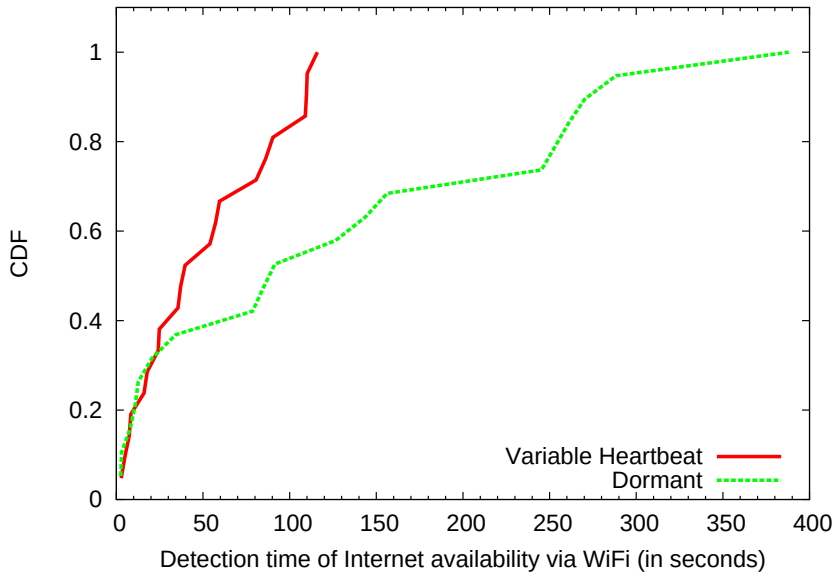

Figure 4.11: Detection time of Internet availability via WiFi (in seconds) for 30 percent unavailability, time interval between the decisions as 120 seconds, and the bursts of Internet unavailability as 40X the time between decisions

simulator has transitioned into On state at 15:39:50, the variable heartbeat algorithm detects availability of Internet connectivity via WiFi at 15:41:07 while the dormant algorithm detects at 15:45:36.

With the 1% Internet unavailability, the simulator transitions into the Off state only for 2 times in 9 hours and the total duration in the Off state is 195 and 174 seconds. While the variable heartbeat algorithm detects the availability of Internet connectivity via WiFi in 18 and 13 seconds, the dormant algorithm takes 4 and 44 seconds. We also observed that at arbitrary times, while switching between networks, DHCP IP address assignment via WiFi fails.

4.2.6.3 Probe overhead

Table 4.1 shows the median number of probes[8] sent by the dormant and the variable heartbeat algorithm run with different unavailability time for 6 hours. As we can see from the table, the dormant algorithm sends much less probes compared to the variable heartbeat algorithm. This is because the dormant algorithm sleeps for a long time, thereby probing less and because of that, it spends more time in the cellular network. Since it is in WiFi network for less time than the variable heartbeat algorithm, the number of probes is less in WiFi as well.

The maximum probes of 148 and 197 sent by the variable heartbeat algorithm while in WiFi and cellular, respectively, is due to the fact that the Internet unavailability in that run was 30% and the Internet connectivity went down a high number of times and the duration of downtimes were also longer. Also, while using WiFi, we send a probe only when the passive monitoring observes any anomalous behavior with the statistics. However, the amount of probe traffic is relatively very small (maximum traffic observed is ∼37KB) when compared to the total traffic sent by the smartphone in a period of 6 hours.

We could further reduce the frequency of probing while in the cellular network by not probing when the smartphone is in idle state (no traffic in and out of smartphone). We leave this for future work.

4.2.6.4 Performance when Internet via WiFi is always available and always not available

When Internet via WiFi is always available, our passive monitoring technique will be in action. With the traffic flowing in and out of the smartphone, we just periodically observe

[8]We send 3 pings per probe. In our experiments, we observe that with good Internet connectivity, sometimes only 1 reply is obtained for 3 ping requests. Hence, we send 3 pings per probe and not less.

Table 4.1: Median number of probes sent for 10, 20 and 30% unavailability with tests run for 6 hours (minimum, maximum in brackets)

Algorithm	WiFi	Cellular
Dormant	29 (22, 42)	81 (33, 148)
Variable Heartbeat	41 (25, 45)	107 (40, 197)

the statistics. But when there is no traffic in and out of the smartphone, we probe a reliable server (once per minute in our experiments). So, in the worst case that the smartphone is completely idle, we probe every minute.

When Internet via WiFi is not available for the entire time, we detect the unavailability and switch to the cellular network. In the cellular network, once we reach the estimated time by adaptively sleeping, we probe every 30 seconds in our experiments. But this situation occurs only when the Internet via WiFi is not available. If there is no WiFi access point in the surroundings or if the smartphone is not able to connect to any access point, our algorithm will not be activated at all.

4.2.7 Energy overhead

We detect the availability of Internet connectivity via WiFi within 60 seconds 90% of the time. Existing solutions do not detect the availability of Internet connectivity via WiFi at all and stay in the cellular network. This leads to an increased energy usage and we give some examples to show how much energy is saved by our approach. We account only for the power used by the network elements.

Based on the power model and measured values from [17], we calculate the power used in the following scenarios.

- Suppose the smartphone is idle for the entire period of time, only beacon messaging or paging will be done. If the smartphone uses LTE, energy spent per minute is 27850 mW. If the smartphone uses WiFi network, energy spent per minute is 15040 mW. Hence, we save a lot more energy than the existing solutions by switching to WiFi whenever its appropriate.

- Suppose the smartphone transmits and receives at a throughput of 1Mbps, the smartphone using LTE uses 1778.4 mW for the data transfer and a base power of 1060 mW to be in the connected state, while the smartphone using WiFi uses 553 mW for the data transfer and base power of 119.3 mW. To calculate these values, we used the power model equation from [17],

$$P = \alpha_u t_u + \alpha_d t_d + \beta \tag{4.5}$$

where, P is the power, t_u is the upload throughput, and t_d is the download throughput. $\alpha_u, \alpha_d, \beta$ are the best fit parameters for the power model given by Table 4 in [17].

- Suppose that the smartphone transmits and receives at the highest throughput possible provided by the carriers (16.65Mbps upload and 7.43Mbps download by AT & T [18]), the smartphone using LTE uses 5410.5 mW for the data transfer and a base power of 1060 mW to be in the connected state, while the smartphone using WiFi uses 4518 mW for the data transfer and base power of 119.3 mW.

Here is an example on how much energy we save over the existing solutions. In Figure 4.12, the simulator stays in the On state for 25 minutes, then transitions into the Off state and stays for 15 minutes before transitioning back into the On state. With the values used in our passive monitoring algorithms, we detect the unavailability of Internet connectivity via WiFi within 60 seconds and hence, at the 26th minute, our algorithm switches the smartphone into the cellular network, while the existing smartphones take more than 5 minutes to detect the unavailability of Internet connectivity via WiFi. The variable heartbeat algorithm detects the availability of Internet connectivity via WiFi within 60 seconds, 95% of the times and hence, we switch to WiFi network at the 41st minute and stay in WiFi while the existing smartphones does not detect the availability of Internet connectivity via WiFi and stays in cellular network.

Table 4.2 shows the energy saved by the variable heartbeat algorithm over the existing solutions. As we observe from Figure 4.12, total time unnecessarily spent in the cellular network by our variable heartbeat algorithm is 1 minute and 19 minutes by the existing solutions. Since the existing solutions spend more time in the cellular network, more energy is consumed. In this example, the variable heartbeat algorithm saves $\sim 2500J$.

Also, we measure the energy overhead for enabling and disabling WiFi while in the cellular network. We take battery measurements using standard Android APIs and observe that, for enabling and disabling WiFi 100 times while being in the cellular network, $\sim 2\%$ of battery is used. As the number of times enabling and disabling WiFi increases, battery usage also increases considerably. For example, for enabling and disabling 500 times, $\sim 14\%$ of battery is used.

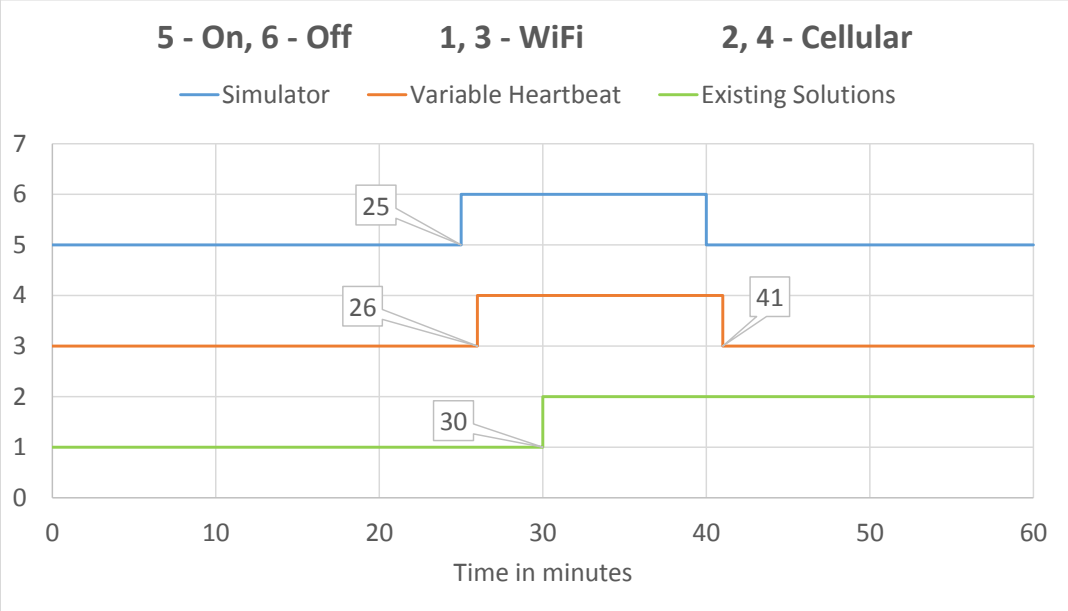

Figure 4.12: Example to demonstrate the energy saved by the variable heartbeat algorithm over existing solutions

Table 4.2: Energy overhead because of extra time spent in cellular network (Assuming smartphones are transmitting and receiving at a constant throughput of 1Mbps)

Algorithm	Time in WiFi after Internet connectivity becomes available	Unnecessary time in cellular network	Energy used
Variable Heartbeat	19 minutes	1 minute	936.3 J
Existing solutions	0 minutes	20 minutes	3405 J

4.2.8 Summary of results

A summary of our results is presented below.

- We appropriately switch between WiFi and cellular networks, thereby providing robust Internet connectivity.
- The variable heartbeat algorithm is better than the dormant algorithm and the existing solutions.
- We are able to detect the availability of Internet connectivity via WiFi faster, while the existing solutions do not do it. Hence, we save a lot of energy.
- The number of probes sent is much less when compared to the traffic sent by the smartphones.

- To our surprise, during the evaluations, we observed a bug with the Android smartphones, where DHCP fails at random instances. Hence, there is no IP address assigned and we might need to retry and/or switch off the interface and turn it on again to acquire an IP address.

CHAPTER 5

CONCLUSION

This thesis aims to address the challenge of providing an efficient and automatic switching between WiFi and cellular networks, when the Internet connectivity beyond the first hop is unavailable. Our initial experiments show that the existing solutions are inefficient and that they do not work well in many scenarios. To detect the unavailability of Internet connectivity via WiFi, we have come up with a solution which utilizes a combination of active and passive monitoring techniques. Once we detect the unavailability and switch to the cellular network, we need to switch back to WiFi as soon as possible, since cellular networks consume higher power. We have designed two algorithms, namely the dormant and variable heartbeat adaptive algorithms, that adaptively check if the Internet connectivity via WiFi is available or not. For probing the WiFi network while in the cellular network, we enable the simultaneous access of both the interfaces. We have built a prototype solution which realizes the algorithms designed that allows an Android smartphone to switch from one type of network to another depending on the status of Internet connectivity through the WiFi network. We have evaluated our algorithms with the Internet availability simulator designed based on the two-state continuous time Markov chain. Our results show that we are able to detect unavailability of Internet connectivity beyond the first hop and switch to the cellular network. With the variable heartbeat algorithm, we are able to switch back to WiFi within 60 seconds of Internet connectivity being available (90% of the time), and we are efficient in probing as well.

REFERENCES

[1] "Worldwide Smartphone Shipments Top One Billion Units for the First Time," http://www.idc.com/getdoc.jsp?containerId=prUS24645514.

[2] "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 20122017," http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.html.

[3] Sarthak Grover, Mi Seon Park, Srikanth Sundaresan, Sam Burnett, Hyojoon Kim, Bharath Ravi, Nick Feamster, "Peeking Behind the NAT: An Empirical Study of Home Networks," IMC13, October 2325, 2013, Barcelona, Spain.

[4] "WiFi Watchdog State Machine-Android Source Code," https://github.com/android/platform_frameworks_base/blob/master/wifi/java/android/net/wifi/WifiWatchdogStateMachine.java.

[5] Ganesh Ananthanarayanan, Ion Stoica, "Blue-Fi: Enhancing Wi-Fi Performance using Bluetooth Signals," MobiSys09, June 22 - 25, 2009, Krakw, Poland.

[6] Lenin Ravindranath, Calvin Newport, Hari Balakrishnan and Samuel Madden, "Improving Wireless Network Performance Using Sensor Hints," NDSL 2011.

[7] "Network Connectivity Status Indicator in Microsoft Windows OS," http://technet.microsoft.com/en-us/library/ee126135(v=ws.10).aspx.

[8] "Concurrent 3G/4G and WiFi for Android," https://sites.google.com/site/lotuseaterarpit/news/3g4gwififorandroidconcurrently.

[9] "Let WiFi and 3G connection work together by hacking ConnectivityService.java," http://mobisocial.stanford.edu/news/2011/03/let-wifi-and-3g-connection-work-together-by-hacking-connectivityservice-java/.

[10] "Enable mobile data programmatically," http://stackoverflow.com/a/13523517/2644071.

[11] "CyanogenMod," http://www.cyanogenmod.org/.

[12] "Busybox," https://code.google.com/p/busybox-android/.

[13] Feng Qian, Zhaoguang Wang, Yudong Gao, Junxian Huang, Alexandre Gerber, Zhuoqing Morley Mao, Subhabrata Sen, and Oliver Spatscheck, "Periodic Transfers in Mobile Applications: Network-wide Origin, Impact, and Optimization," WWW 2012, April 1620, 2012, Lyon, France.

[14] "Measuring Broadband America - FCC," http://www.fcc.gov/measuring-broadband-america.

[15] Jun Cheol Park, Sneha Kumar Kasera, "Expected Data Rate: An Accurate High-Throughput Path Metric For Multi-Hop Wireless Routing," Secon 2005.

[16] Jorg Nonnenmacher, Ernst W. Biersack, and Don Towsley, "Parity-Based Loss Recovery for Reliable Multicast Transmission," IEEE/ACM Transactions on networking, vol. 6, no. 4, August 1998.

[17] Junxian Huang, Feng Qian, Alexandre Gerber, Zhuoqing Morley Mao, Subhabrata Sen, and Oliver Spatscheck, "A Close Examination of Performance and Power Characteristics of 4G LTE Networks," MobiSys12, June 2529, 2012, Low Wood Bay, Lake District, UK.

[18] "AT&T's 4G LTE network found to be the fastest in the U.S," http://bgr.com/2013/06/17/4g-lte-speeds-att-verizon-sprint-t-mobile/.